Chap. 8

Architectural Support for System Development

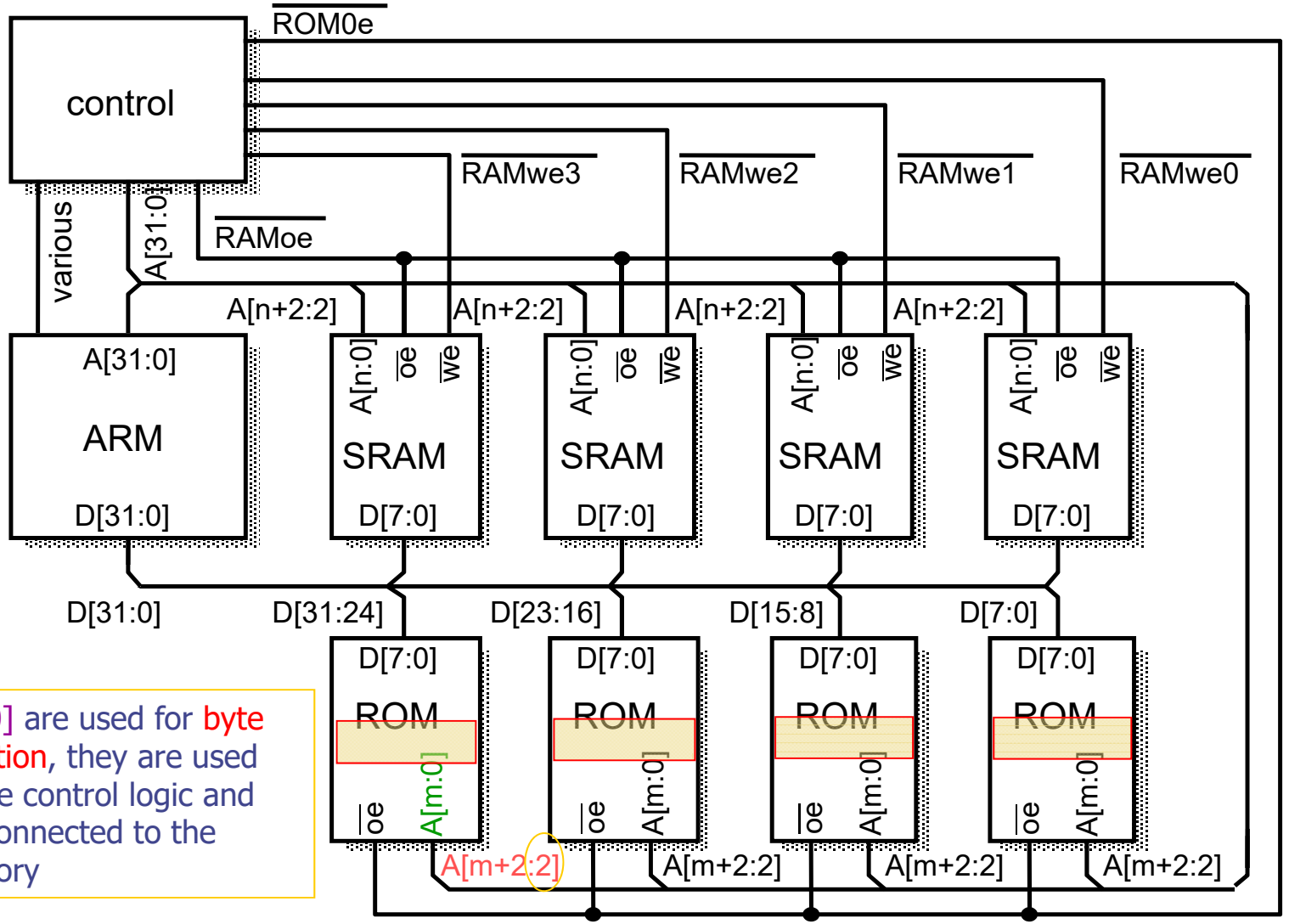
8.1 The ARM memory interface

- ARM memory interface
  - Connect an ARM processor to a memory system
  - More recent ARM cores have direct AMBA interfaces (Section 8.2)
  - Many basic issues discussed here
- ARM bus signals
  - Memory bus interface signals include the following
    - A 32-bit address bus, $A[31:0]$, which gives the byte address of the data to be accessed
    - A 32-bit bidirectional data bus, $D[31:0]$, along which the data is transferred
    - Signals that specify whether the memory is needed ($\overline{mreq}$) and whether the address is sequential (seq)
      - These are issued in the previous cycle so that the memory control logic can prepare appropriately
    - Signals that specify the direction ($\overline{r}/w$) and size ($\overline{b}/w$ or $mas[1:0]$) of the transfer
    - Bus timing and control signals

8.1 The ARM memory interface

- Simple memory interface
  - The simplest form of memory interface is suitable for operation with ROM and static RAM (SRAM)
  - Fig. 8.1
- Control logic
  - The control logic performs the following functions
    - decide when to activate the RAM and when to activate the ROM
      - *A*[31] is low ⇒ ROM, *A*[31] is high ⇒ RAM
      - Remap is often required => exception vectors, access speed
    - control the byte write enables during a write operation
      - byte, half-word, word
    - ensure that the data is ready before the processor continues
      - set the clock to suit RAM accesses and use wait states for (typically slower) ROM and peripheral accesses
  - Fig. 8.2

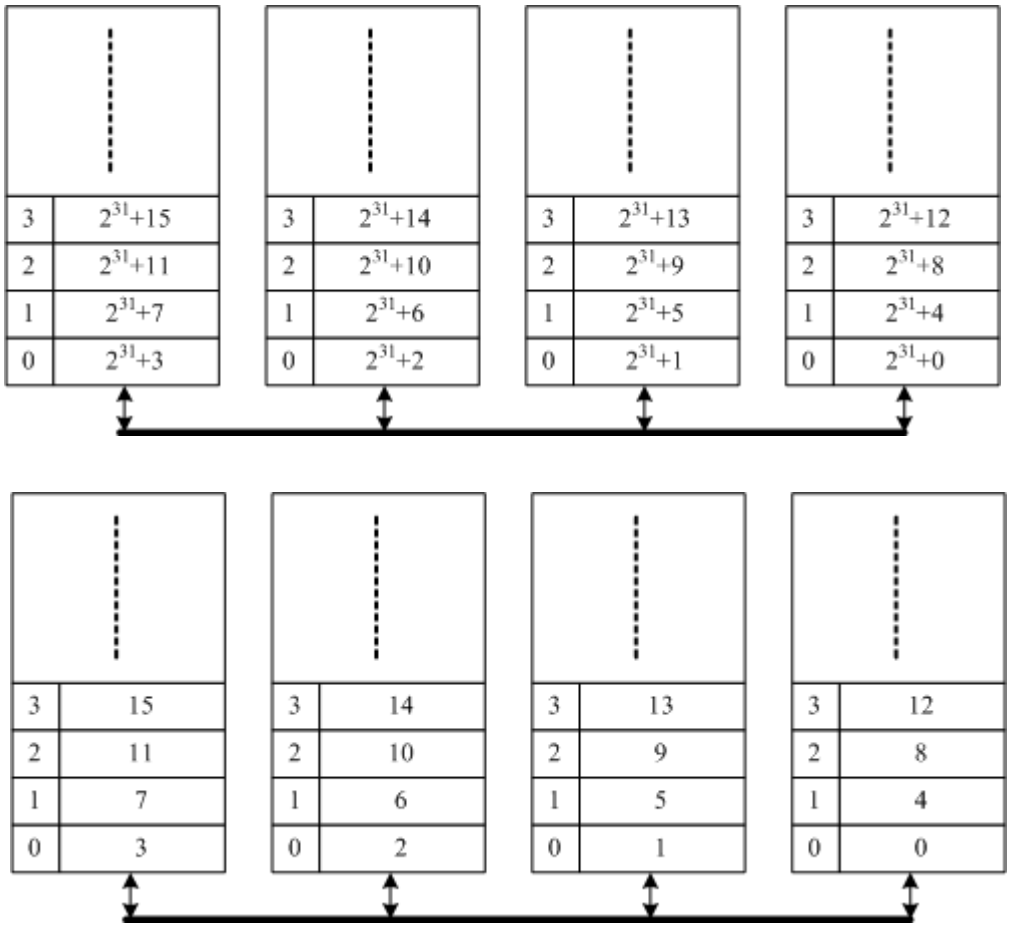
Fig. 8.1 A basic ARM memory system



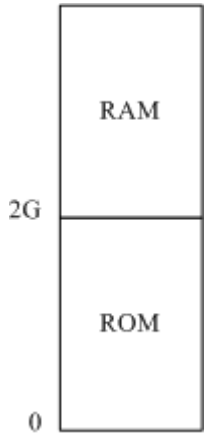
A[1:0] are used for byte selection, they are used by the control logic and not connected to the memory

128 Kbyte ROM ⇒ 2^{17}, *A*[*16:0*] ⇒ *A*[*18:2*] (4ROMs)

Fig. 8.1 A basic ARM memory system

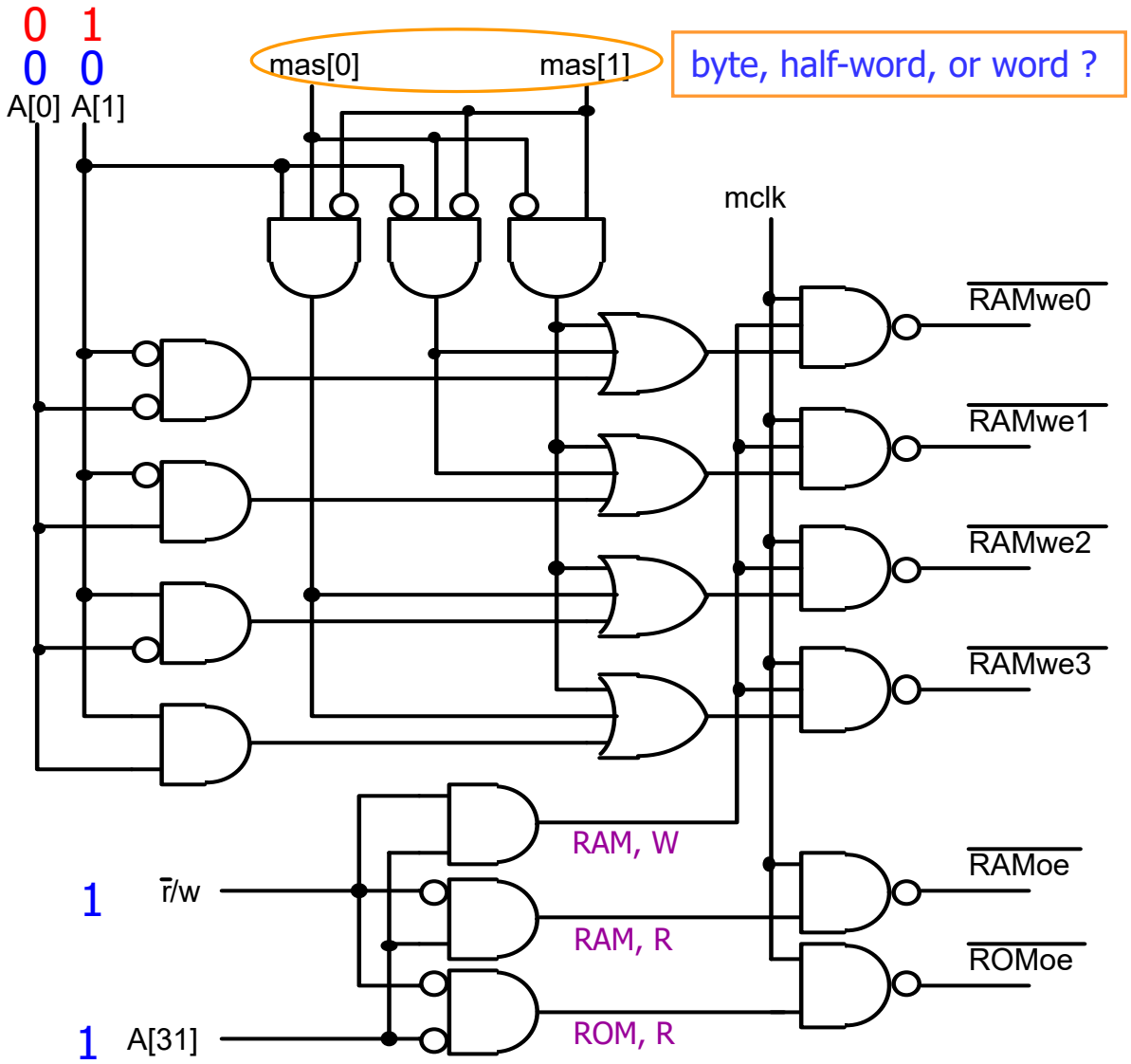


RAM address space



ROM address space

Fig. 8.2 Simple ARM memory system control logic



mas[0]	mas[1]	
0	0	byte
0	1	word
1	0	half-word
1	1	x x

8.1 The ARM memory interface

- Wait states (Fig. 8.3, Fig. 8.4, Fig. 8.5)
  - ROM access time: 4 clock cycles

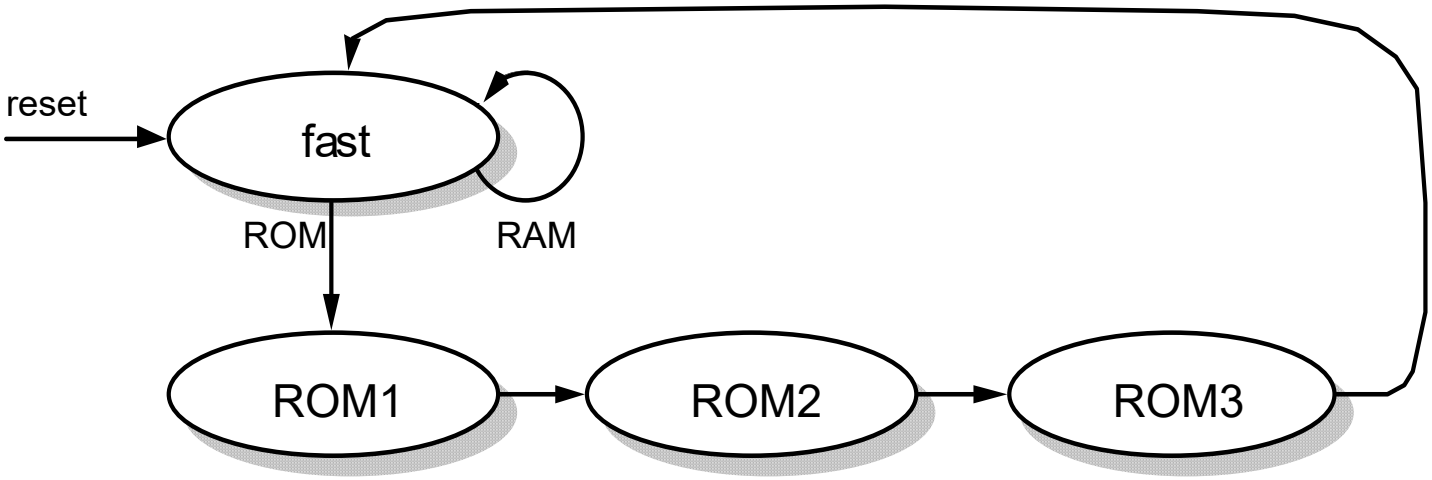


Fig. 8.3 ROM wait control state transition diagram

Fig. 8.4 ROM wait state generator circuit

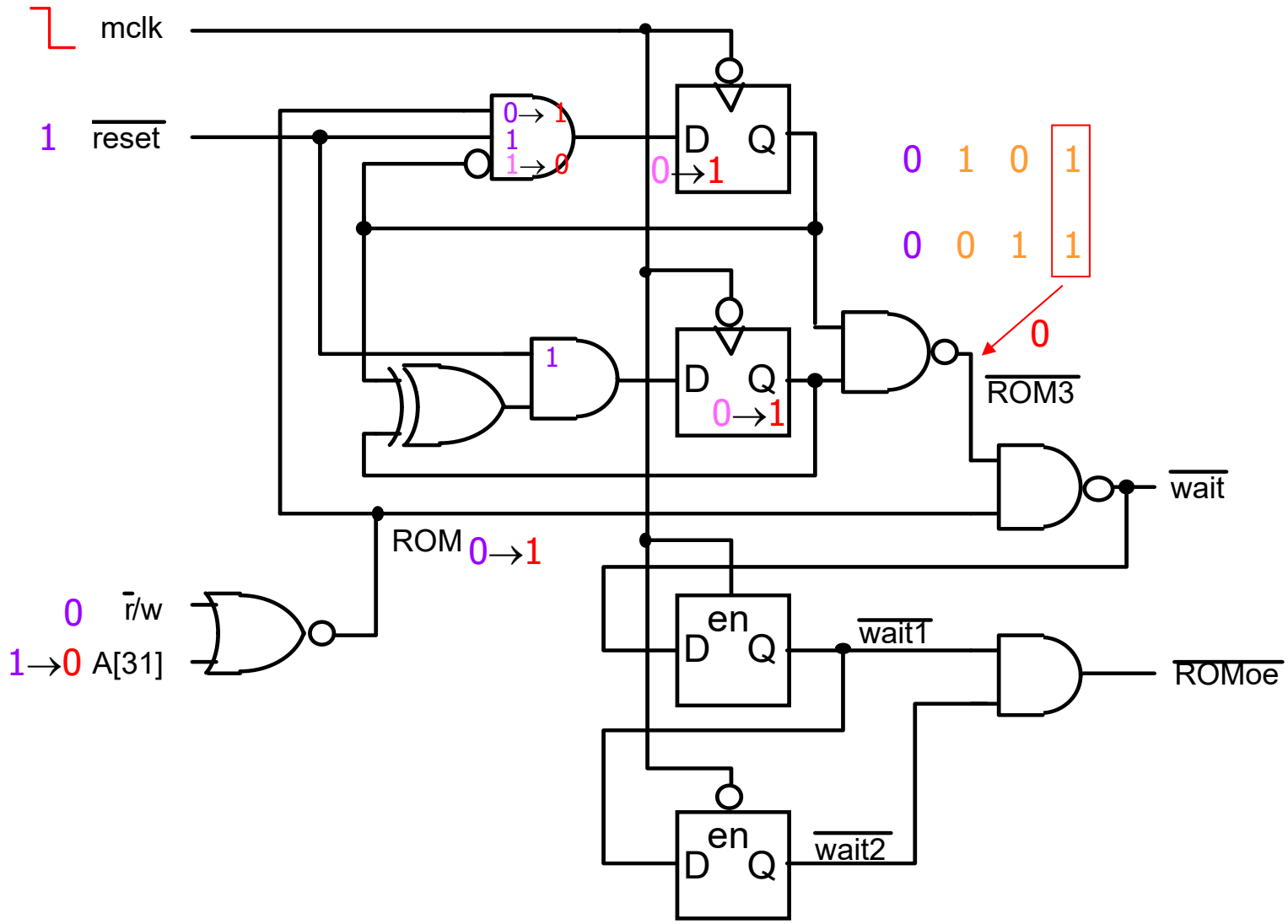
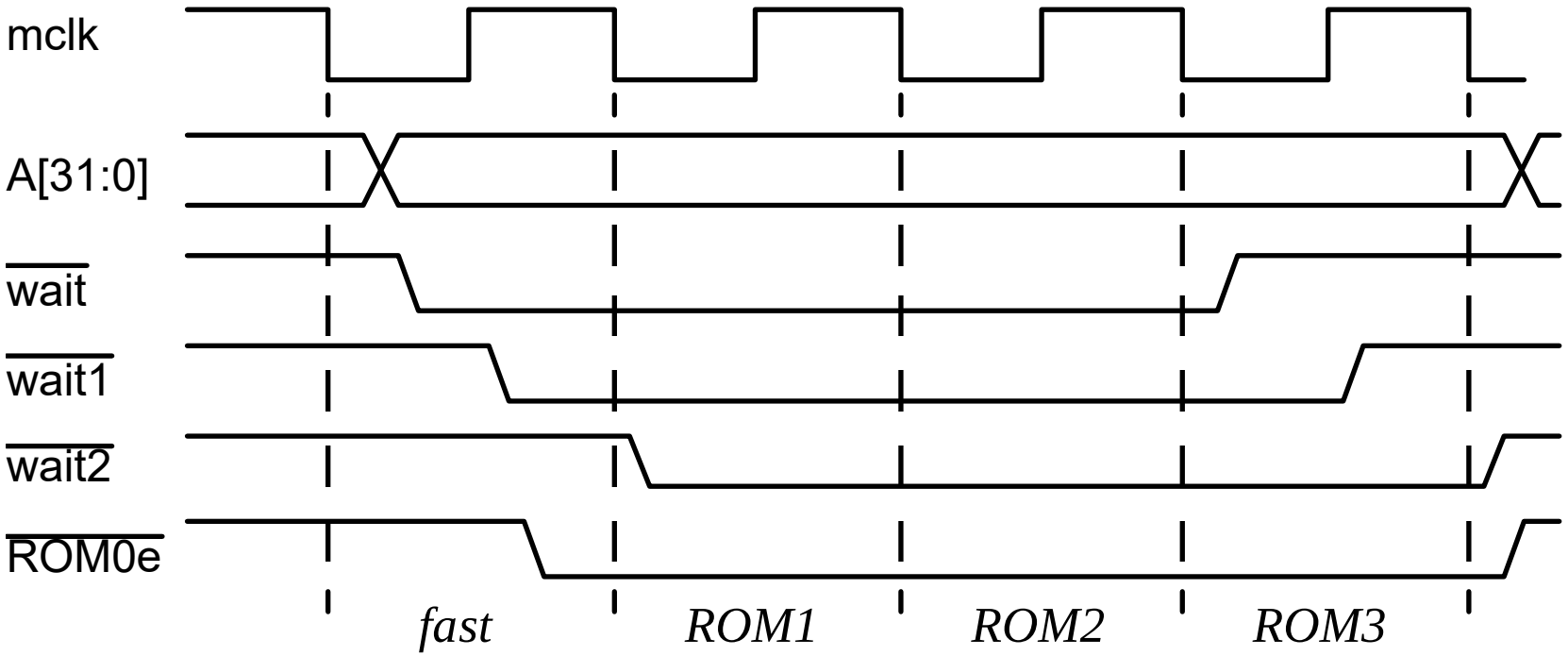


Fig. 8.5 The timing diagram for the ROM wait state logic



8.1 The ARM memory interface

- Sequential accesses
  - If the system is to operate even faster it may not be possible to decode a new address and perform a RAM access in a single clock cycle
  - An extra cycle can be inserted whenever an unknown address (non-sequential) is issued to allow time for address decoding
  - The address that are not unknown are sequential ones
    - these represent around 75% of all addresses in a typical program
  - Fig. 8.6
- DRAM
  - dynamic random access memory
  - Fig. 8.7
    - $\overline{cas}$-only access does not activate the cell matrix it can deliver its data two to three times faster than a full $\overline{ras\text{-}cas}$ access

Fig. 8.6 State transition diagram with a wait state for address decoding

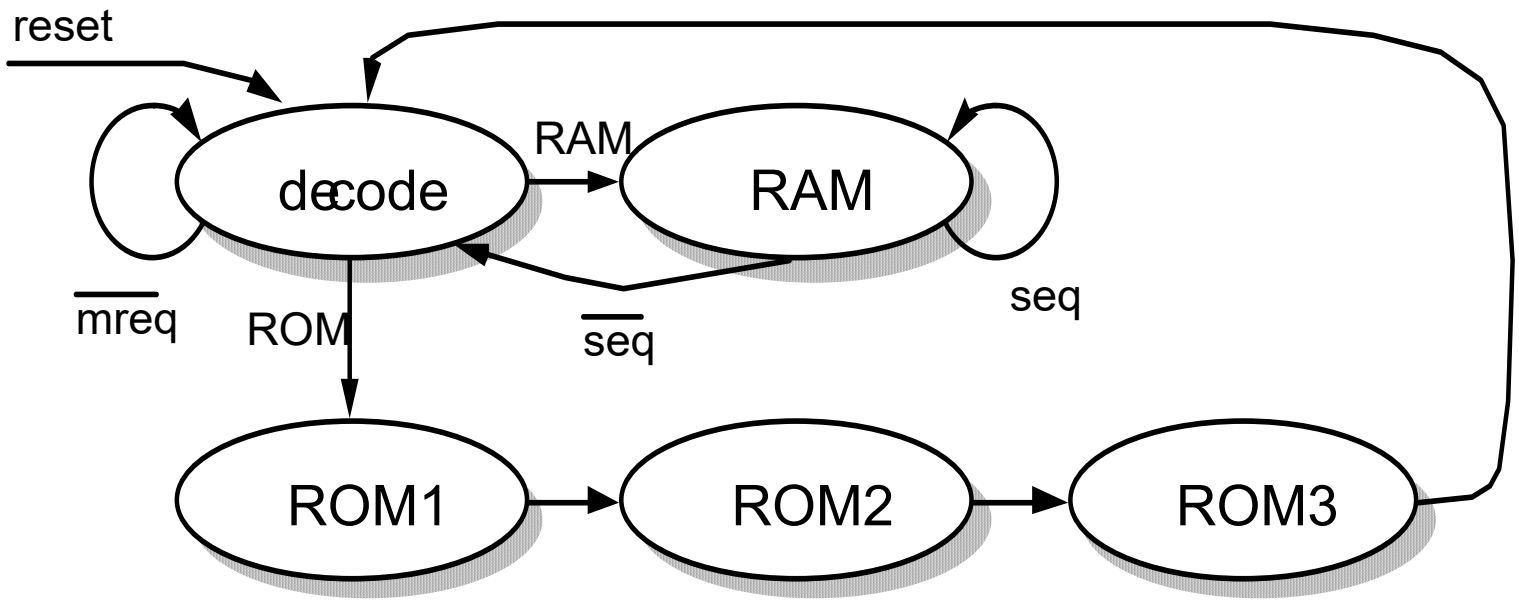
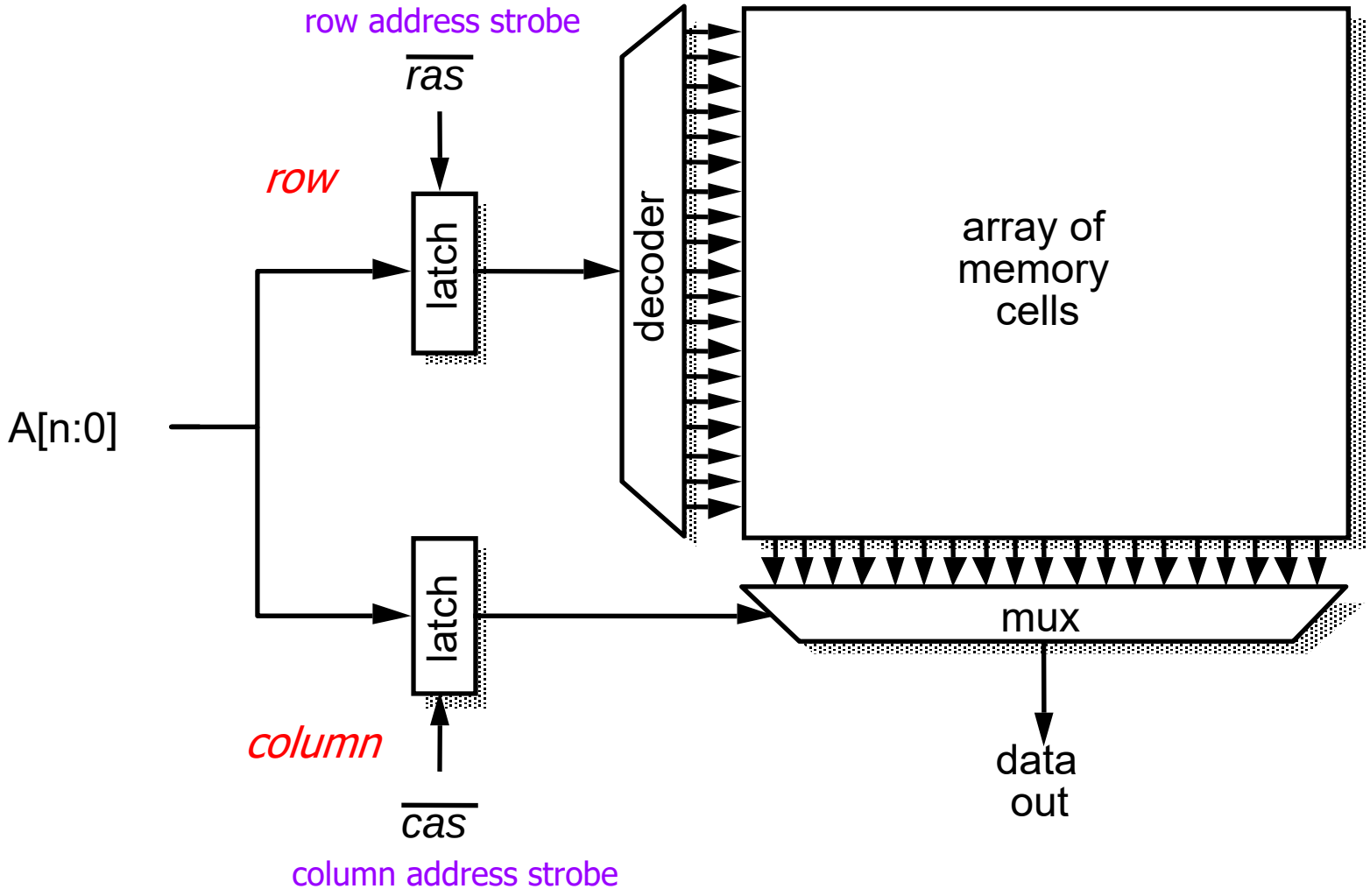


Fig. 8.7 DRAM memory organization



8.1 The ARM memory interface

- ARM address incrementer
  - The solution adopted on the ARM exploits the fact that most addresses (typically 75%) are generated in the address incrementer
  - The ARM address selection logic (Fig. 8.8)
  - A typical DRAM timing diagram is shown in Fig. 8.9
    - first, non-sequential access takes two clock cycles as the row address is strobed in
    - subsequent sequential addresses use a $\overline{cas}$-only access and operate in a single clock cycle

Fig. 8.8 ARM address register structure

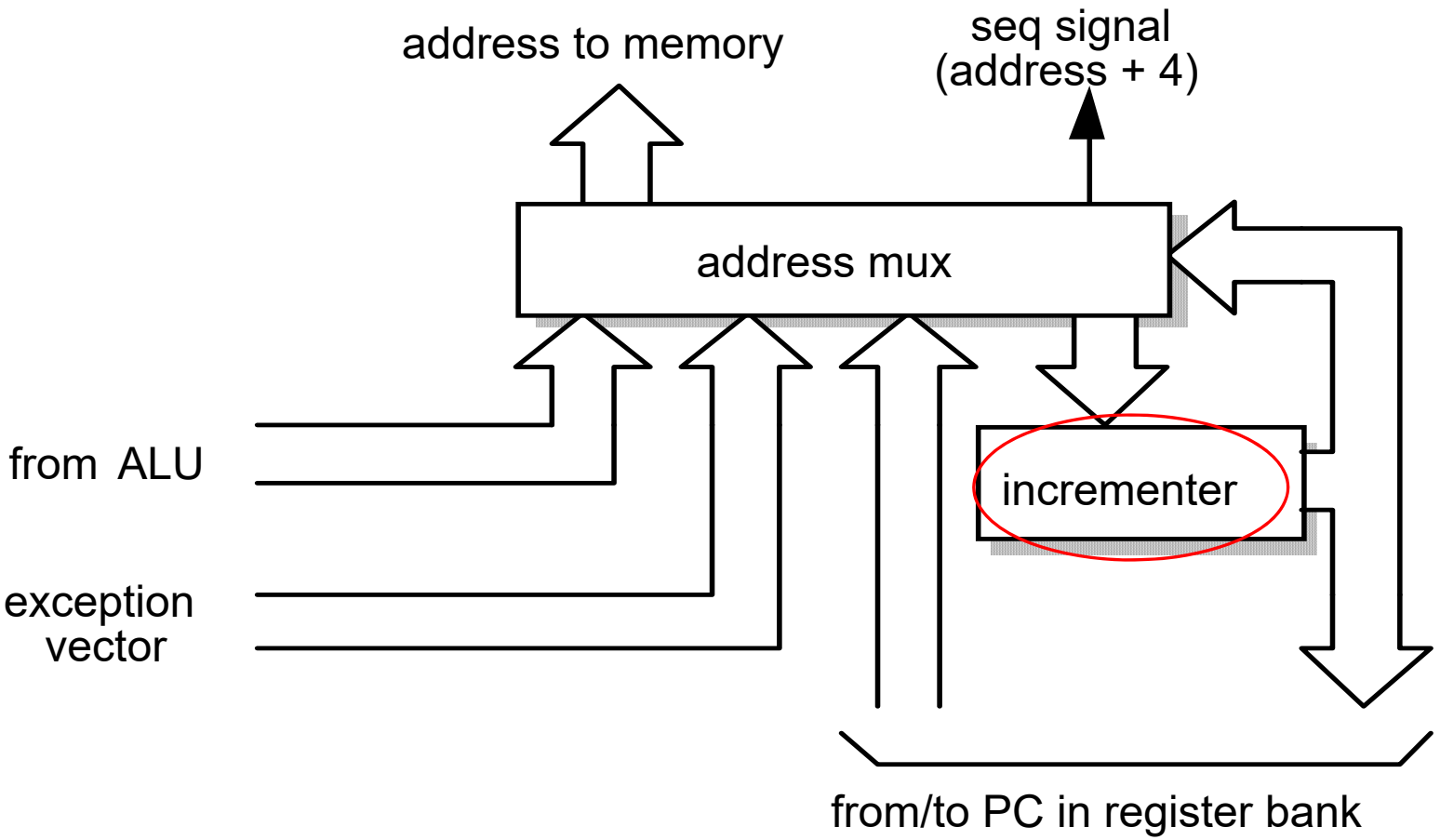


Fig. 8.9 DRAM timing illustration

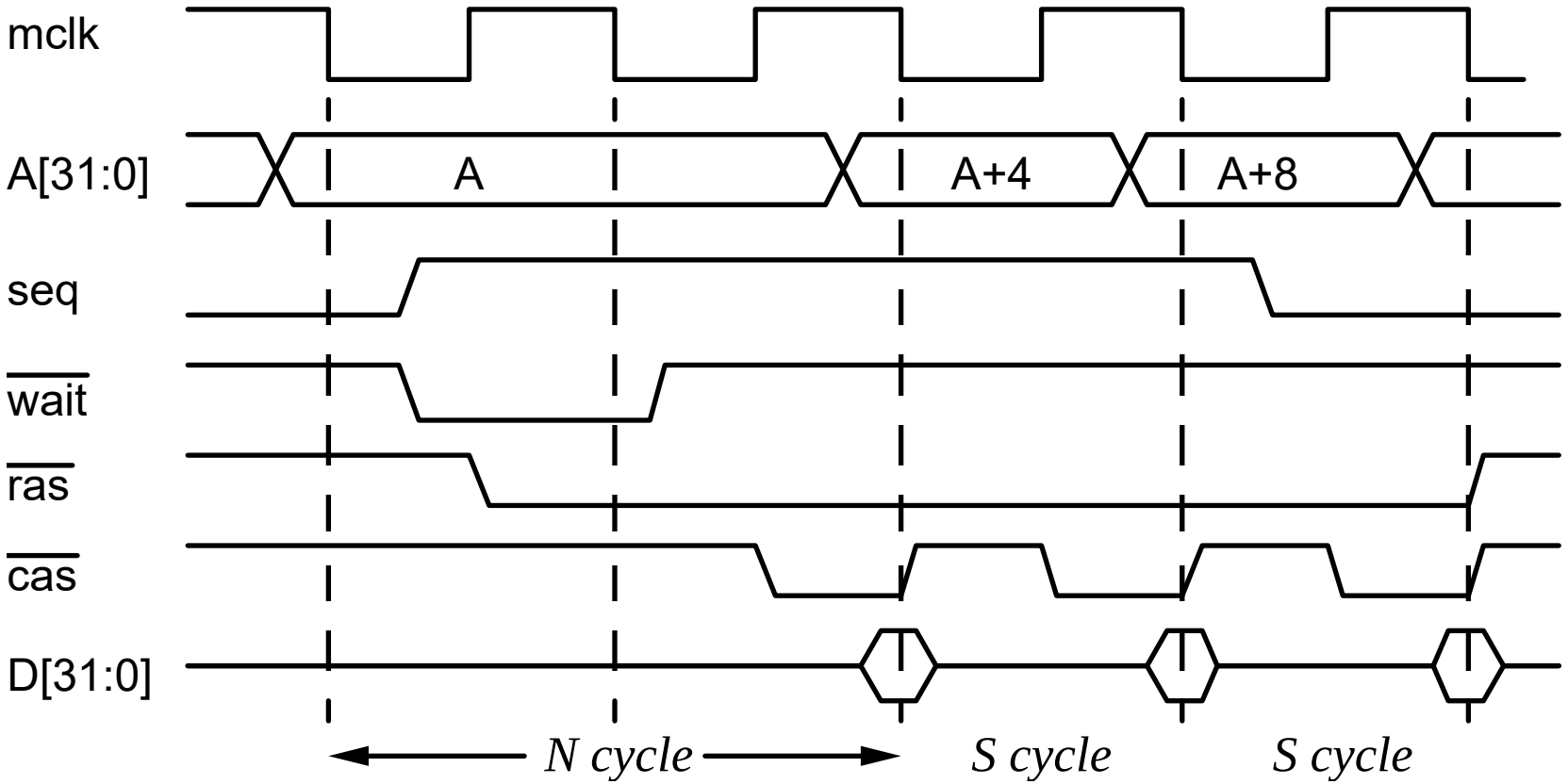
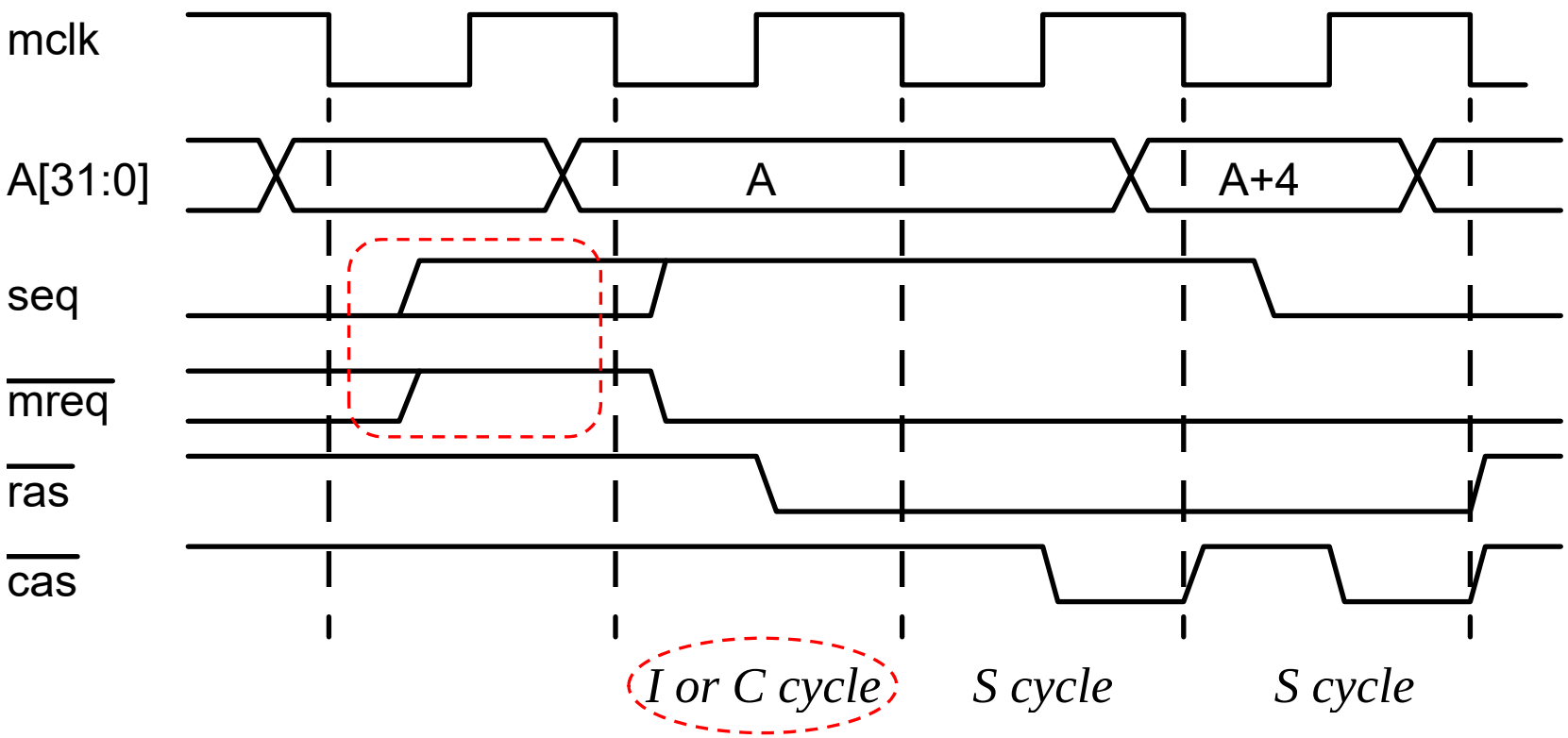


Fig. 8.10 DRAM timing after an internal cycle



ARM7TDMI Cycle Types

$\overline{\text{mreq}}$	seq	Cycle	Use
0	0	N	Non-sequential memory access
0	1	S	Sequential memory access
1	0	I	Internal cycle – bus and memory inactive
1	1	C	Coprocessor register transfer – memory inactive

Memory Layout for Typical ARM Embedded System

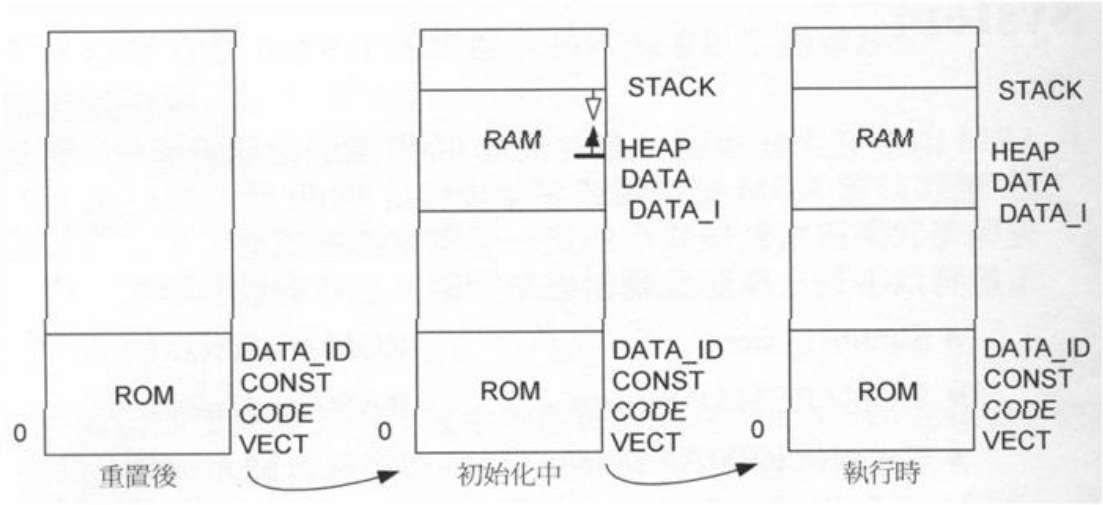
- ARM CPU在系統重置後會從位址0000處開始執行指令
  - 因此在位址0000的“區段”應該放置ROM記憶體或者至少位址0000應該是一組ROM Code，這樣才能確保每次系統啟動後執行的是一個確定的程式碼來完成之後的系統初始化工作。
- 三種記憶體組態
  - Simple System
    - ROM at address 0
  - Simple REMAP System
    - RAM at address 0
  - Complex REMAP System
    - RAM at address 0

Simple System

VECT	中斷向量區段
CODE	程式碼區段
CONST	常數區段
CODE_I	被初始化的程式碼區段(在RAM)。
CODE_ID	此區資料將被”複製”到RAM而形成CODE_I區段。
DATA	資料或變數區段。(在RAM)
DATA_I	被初始化的DATA區段。
DATA_ID	此區資料將被複製到RAM而形成DATA_I區段。
HEAP	動態產生之資料區段，例如 malloc…
STACK	系統及中斷之堆疊區段。

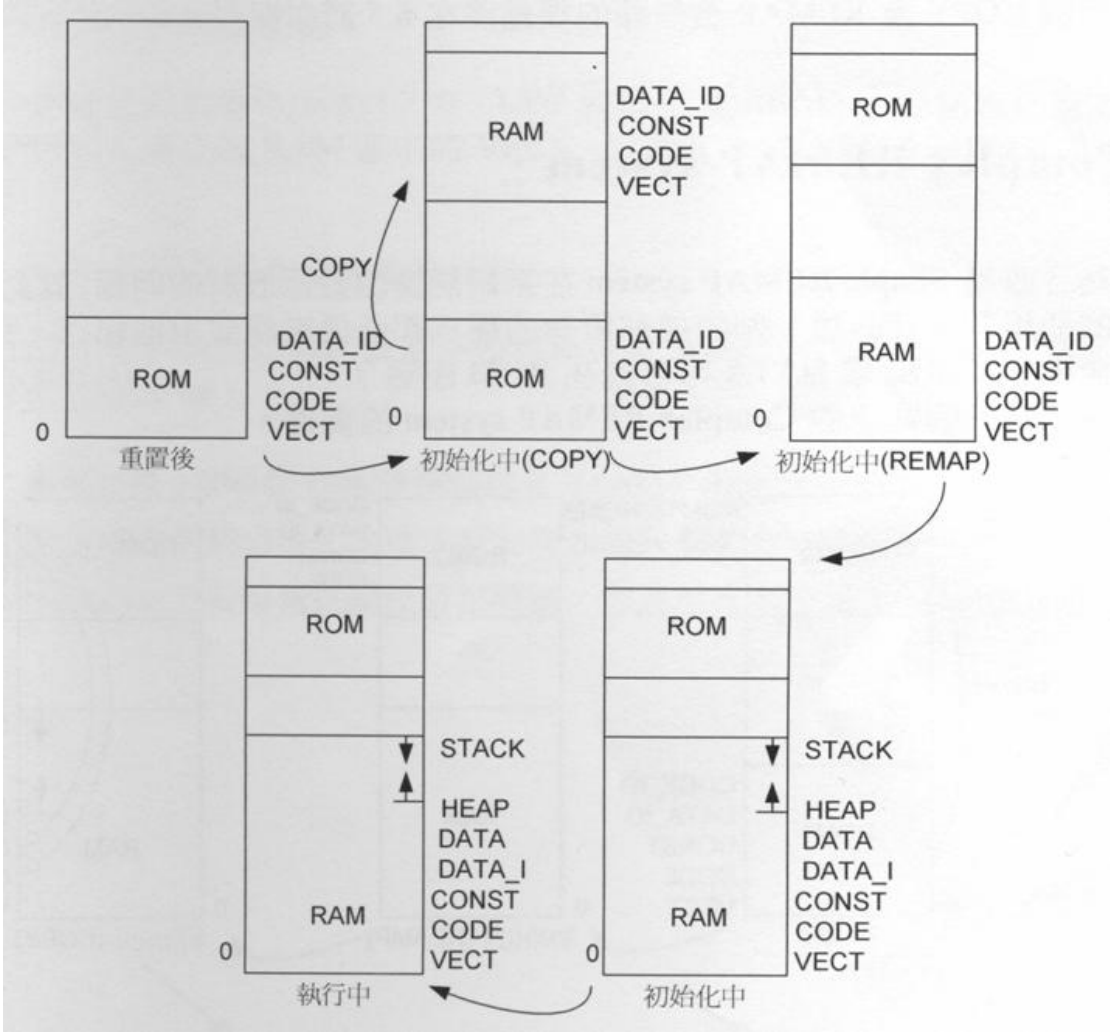
Simple System

- ROM一直是在位址0000的區塊，內含VECT\CODE\CONST及DATA_ID之區段資料
- RAM在初始化的過程中被“啟動”的，DATA_I\HEAP及STACK也在這個階段規劃出來。在執行時期，ROM及RAM之組態保持不變，如果你的應用程式是C的話，main()程序將被執行。
- 缺點
  - ROM的存取速度較慢，程式都在ROM區執行則效率會大打折扣。
  - ROM一般都是8或16bit的晶片，在32bit就要使用2個或4個晶片，這對系統體積大小是一個不利的因素。
  - 無法支援VECT區作動態性的修改



Simple REMAP system

- Creator 採用的是此種記憶體的組態



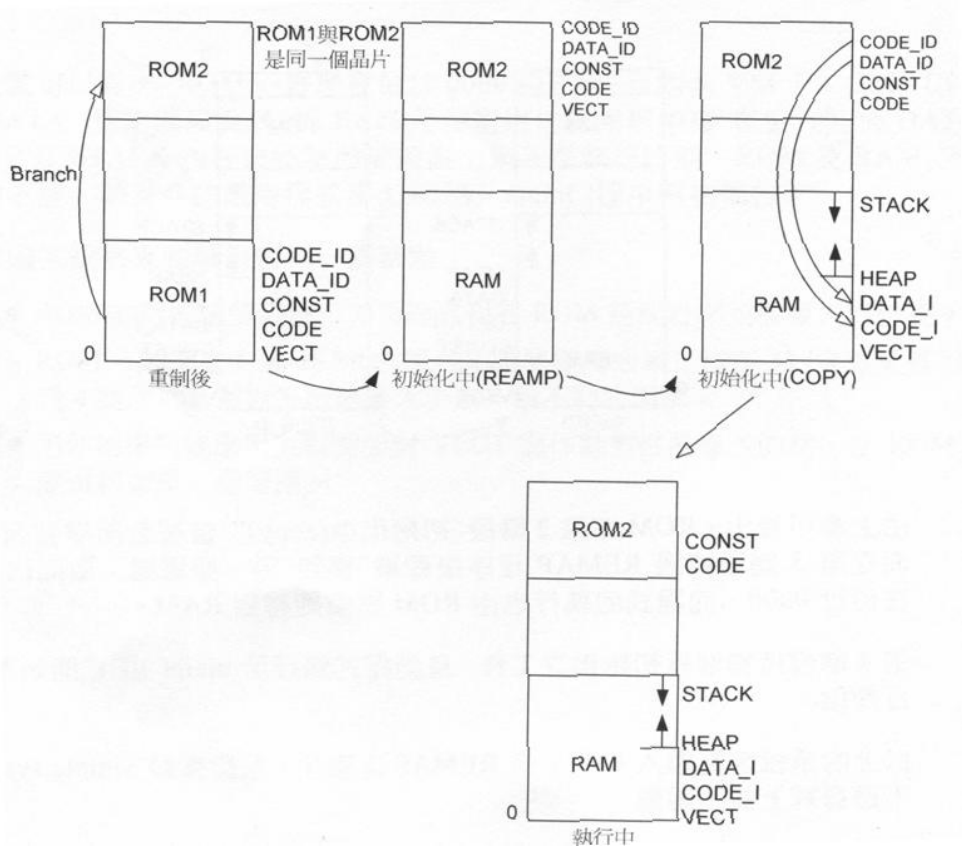
Simple REMAP system

- ROM在第2階段”初始化中(copy)”複製到RAM，在第3階段經過REMAP程序”移到”另一個區域，取而代之的是RAM在位址0000，而程式的執行也由ROM區瞬間移到RAM。
- 第4階段持續進行初始化，直到執行至main()應用程式之執行。
- 由於ROM只在開機重置時扮演COPY的角色，因此無效率上考慮，可以使用慢速及8或16bit的晶片都可勝任。
- 解決了Simp1e system在效率及體積上的缺點，但也有以下之問題:
  - 系統記憶體資源的浪費，因為ROM在執行時期是沒有用的。

Complex REMAP system

- 中ROM1及ROM2看來雖有2塊，但實際上是同一塊晶片。目的提供系統重置ARM CPU在0000位址有ROM code可執行。

- 一般的做法則是在此放一條跳躍指令，由ROM1跳到ROM2之後進行 REMAP之工作而將ROM1隱藏起來，同時再啟動RAM區來取代ROM1所在之位置。在第3階段的初始化(copy)之中，CPU執行的是ROM2中的初始化程式碼，其任務是逐步地建立起RAM區中的VECT\CODE_I\DATA _I\HEAP及STACK區段。而main()的程序將在隨後被執行到。



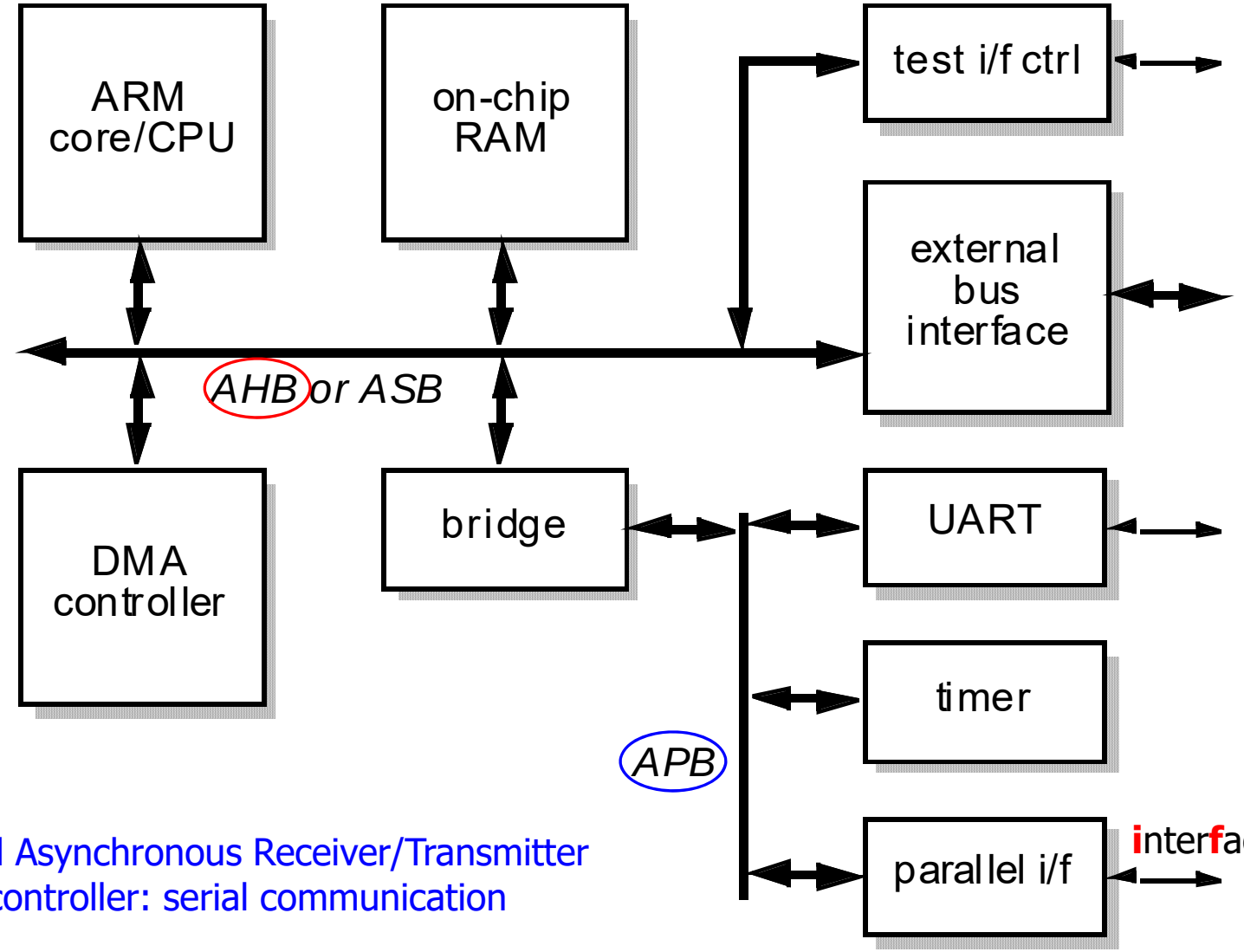
8.2 The Advanced Microcontroller Bus Architecture (AMBA)

- AMBA
  - ARM processor cores have bus interfaces that are optimized for high-speed cache interfacing
  - Some interfacing is required to allow the ARM to communicate with other on-chip macrocells
  - ARM Limited specified the Advanced Microcontroller Bus Architecture, AMBA, to standardize the on-chip connection of different macrocells
- AMBA buses
  - Three buses are defined within the AMBA specification
    - The Advanced High-performance Bus (AHB)
      - connect high-performance system modules, support burst mode data transfers and split transactions, all timing is reference to a single clock edge
    - The Advanced System Bus (ASB)
    - The Advanced Peripheral Bus (APB)
      - offer a simpler interface for low-performance peripherals

8.2 The Advanced Microcontroller Bus Architecture (AMBA)

- ARMA buses (Cont.)
  - A typical AMBA-based microcontroller will incorporate either an AHB or an ASB together with an APB (Fig. 8.11)
  - In the following sections we assume the system bus is an ASB
- Arbitration
  - A bus transaction is initiated by a bus master which requests access from a central arbiter
  - The arbiter decides priorities when there are conflicting requests
  - The ASB only specifies the protocol which must be followed
    - The master, x, issues a request ($AREQx$) to the central arbiter
    - When the bus is available, the arbiter issues a grant ($AGNTx$) to the master

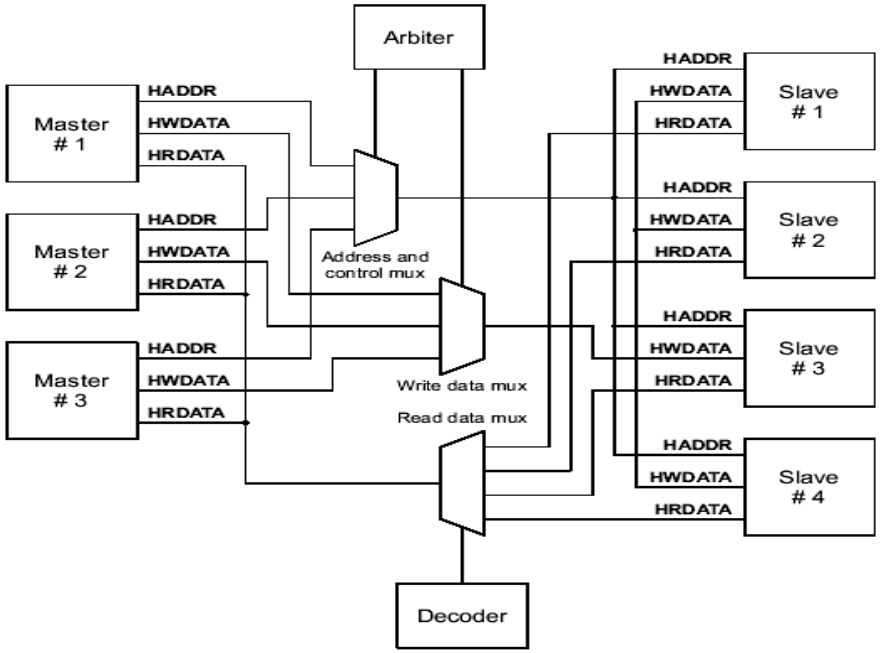
Fig. 8.11 A typical AMBA-based system



Universal Asynchronous Receiver/Transmitter (UART) controller: serial communication

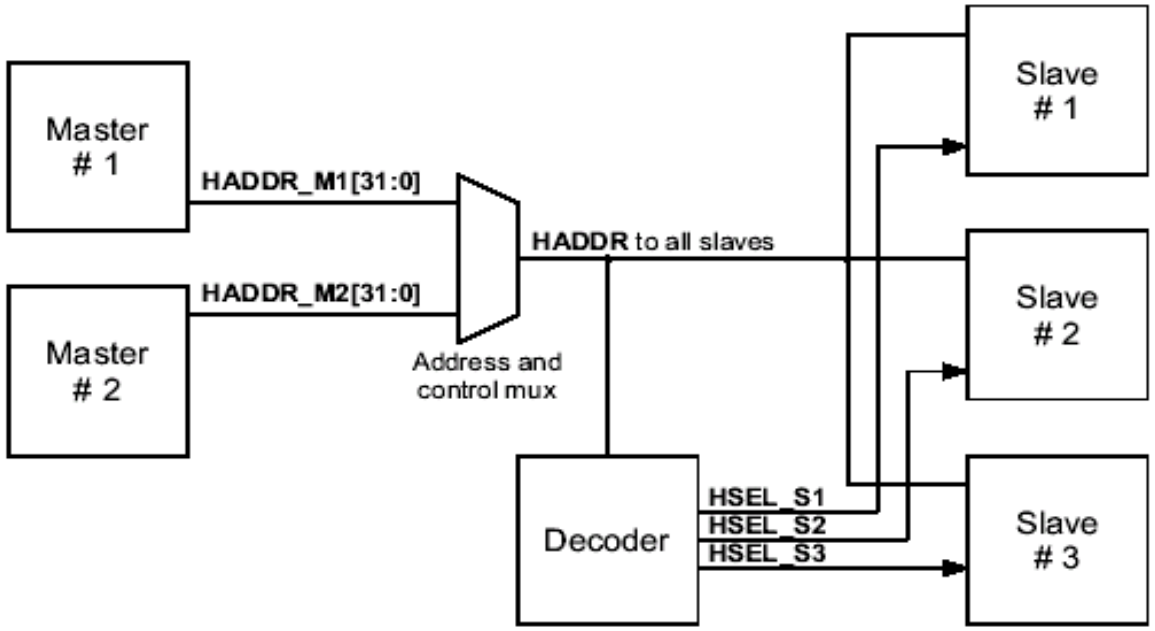
AHB interconnection

- **Bus master drives the address and control**
- **Arbiter selects one of the masters**



Address decoding

- A central address decoder provides HSELx for each slave
- Minimum address space that can be allocated to a single slave is 1K Byte



AHB bus master

- **Initiate read and write by providing an address and control interface**
- **Processor, DMA, DSP test interface**

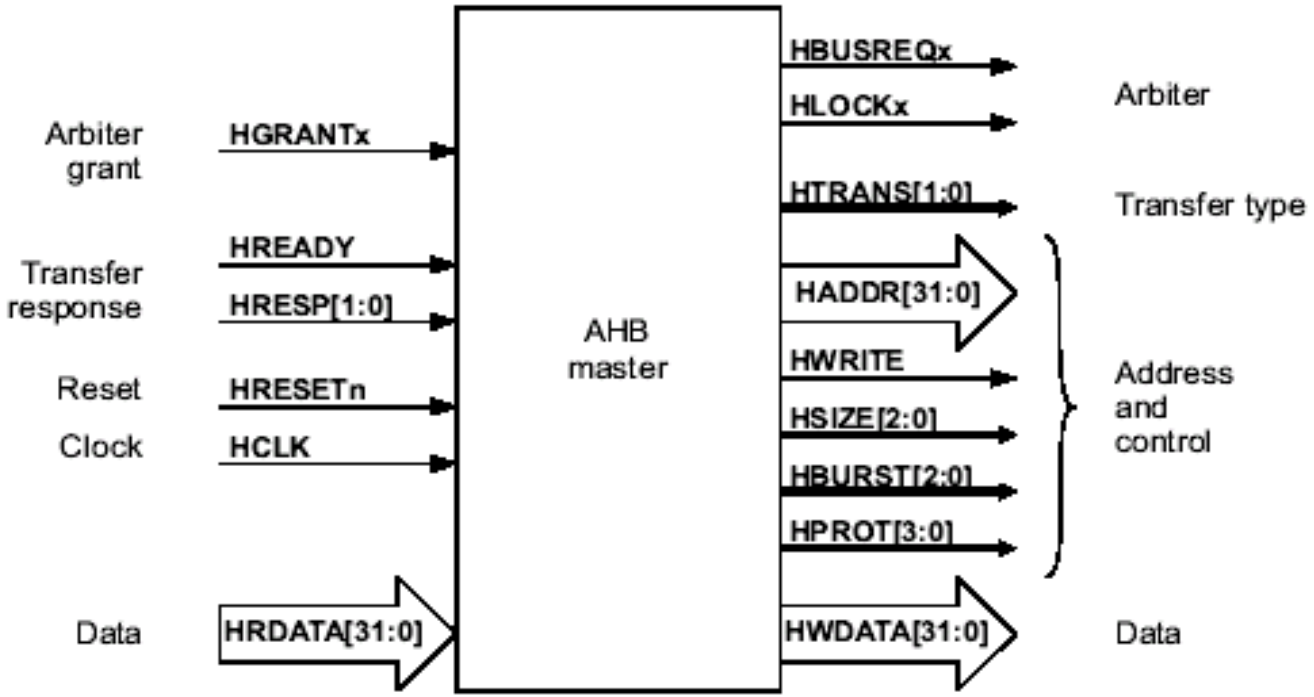
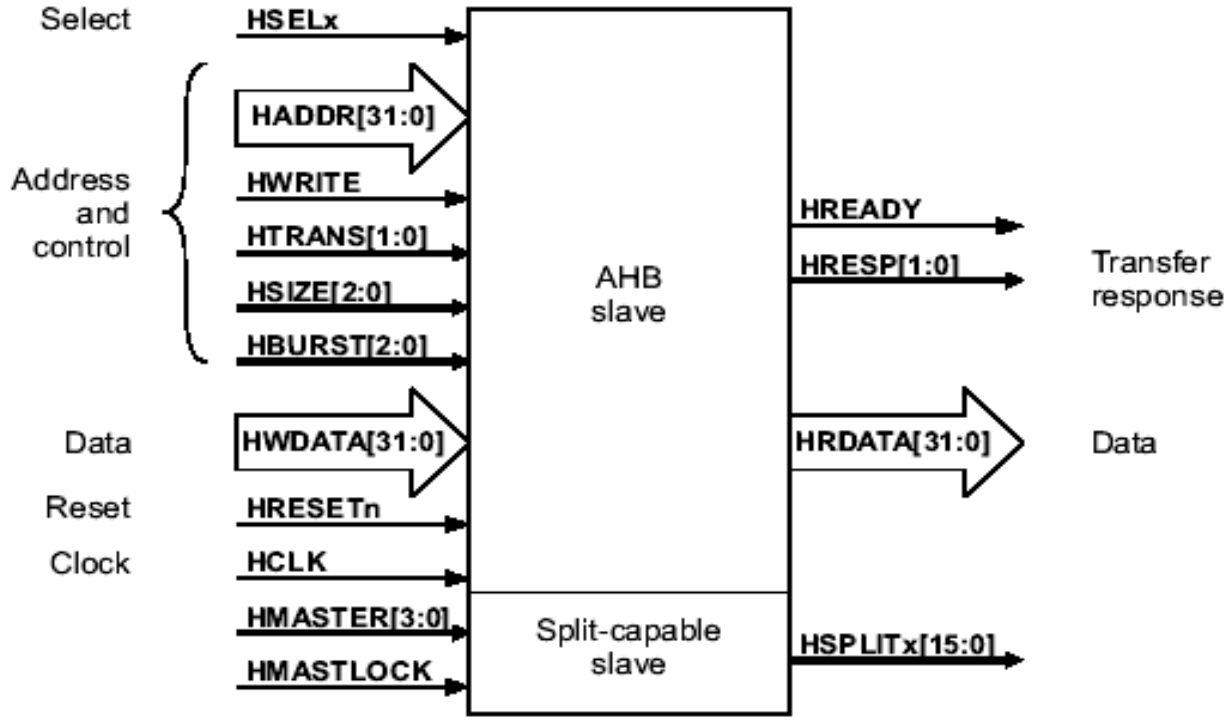


Figure 3-27 AHB bus master interface diagram

AHB bus slave

- **Respond to a read or write operation within a given address-space range**
- **Back to the master the success, failure or waiting**



AHB arbiter

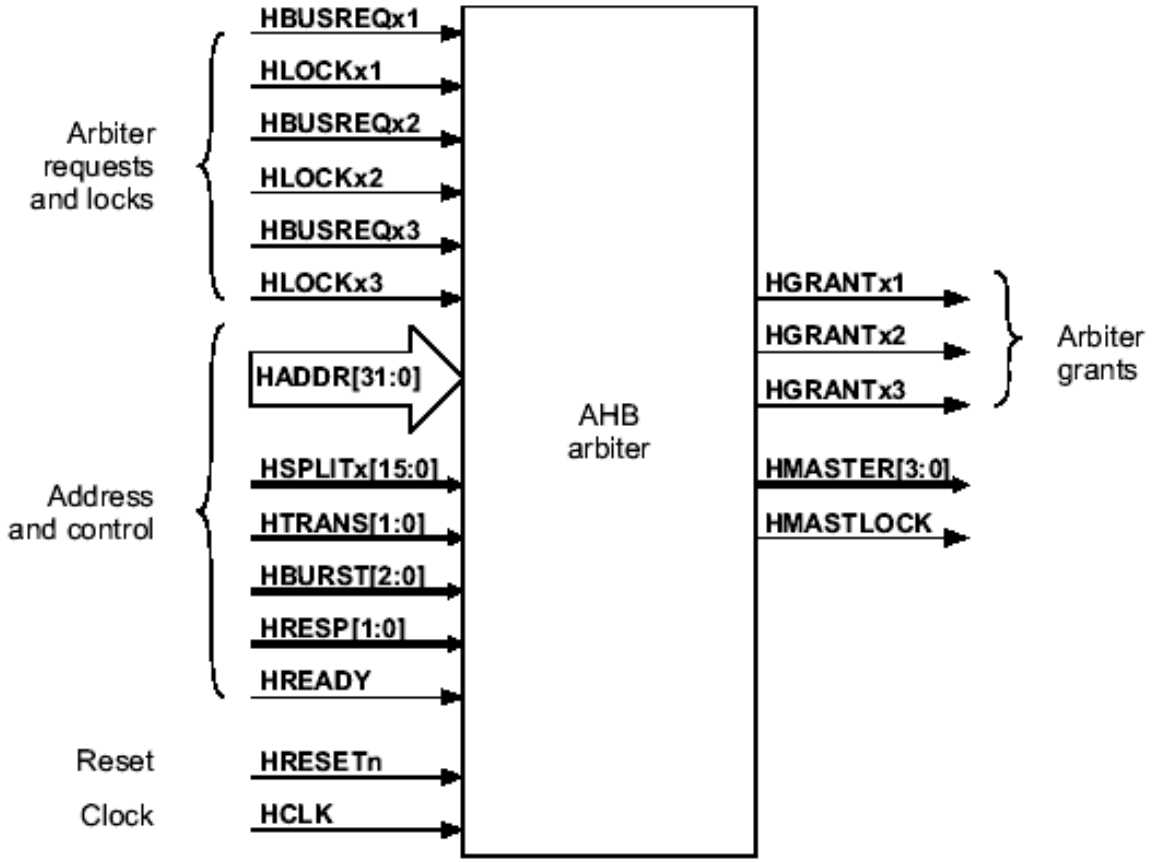
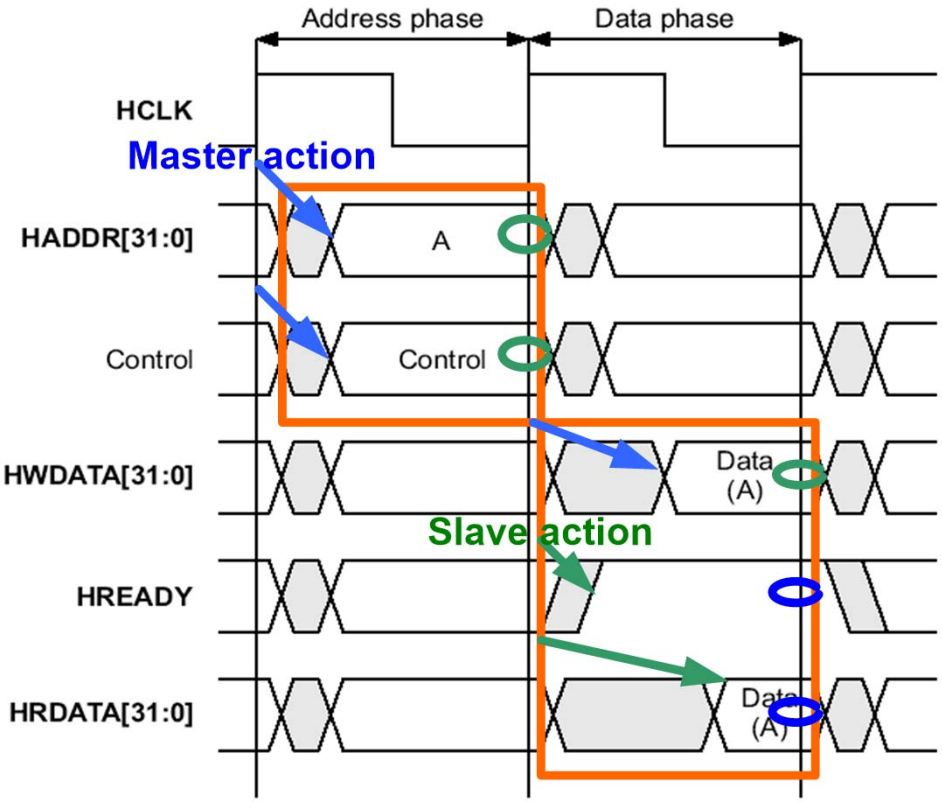
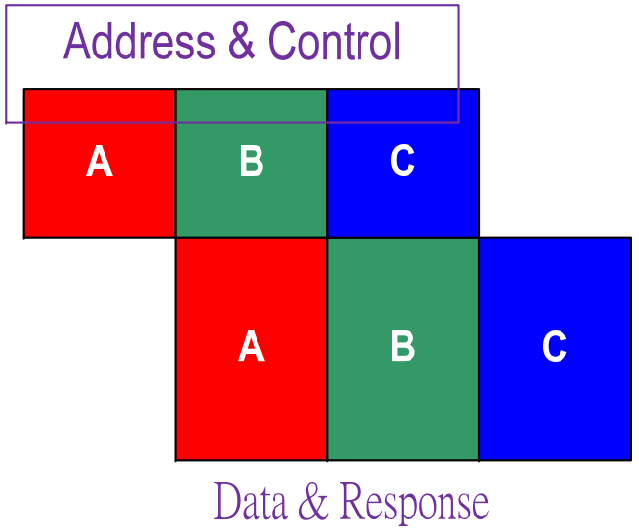


Figure 3-31 AHB arbiter interface diagram

Basic transfer

- **An AHB transfer consists of two distinct sections:**





Transfer with wait state

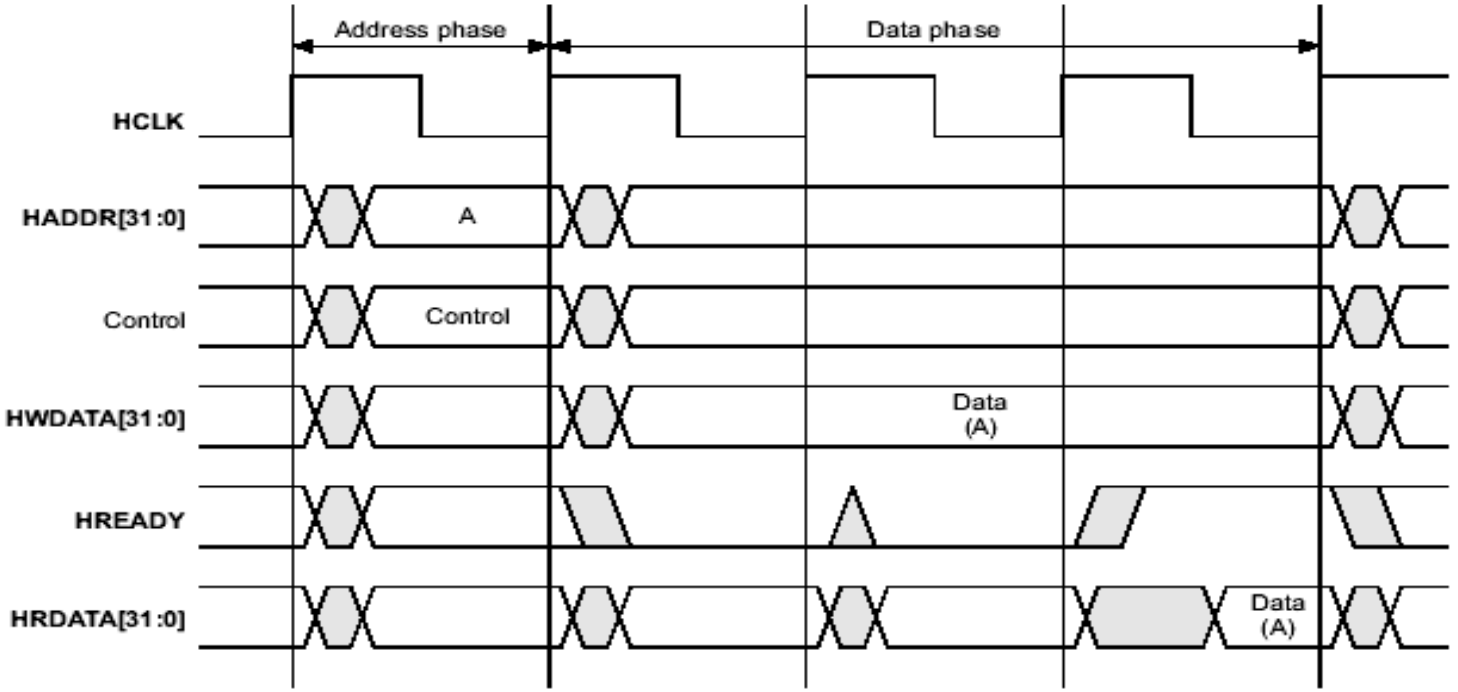


Figure 3-4 Transfer with wait states

Multiple Transfer

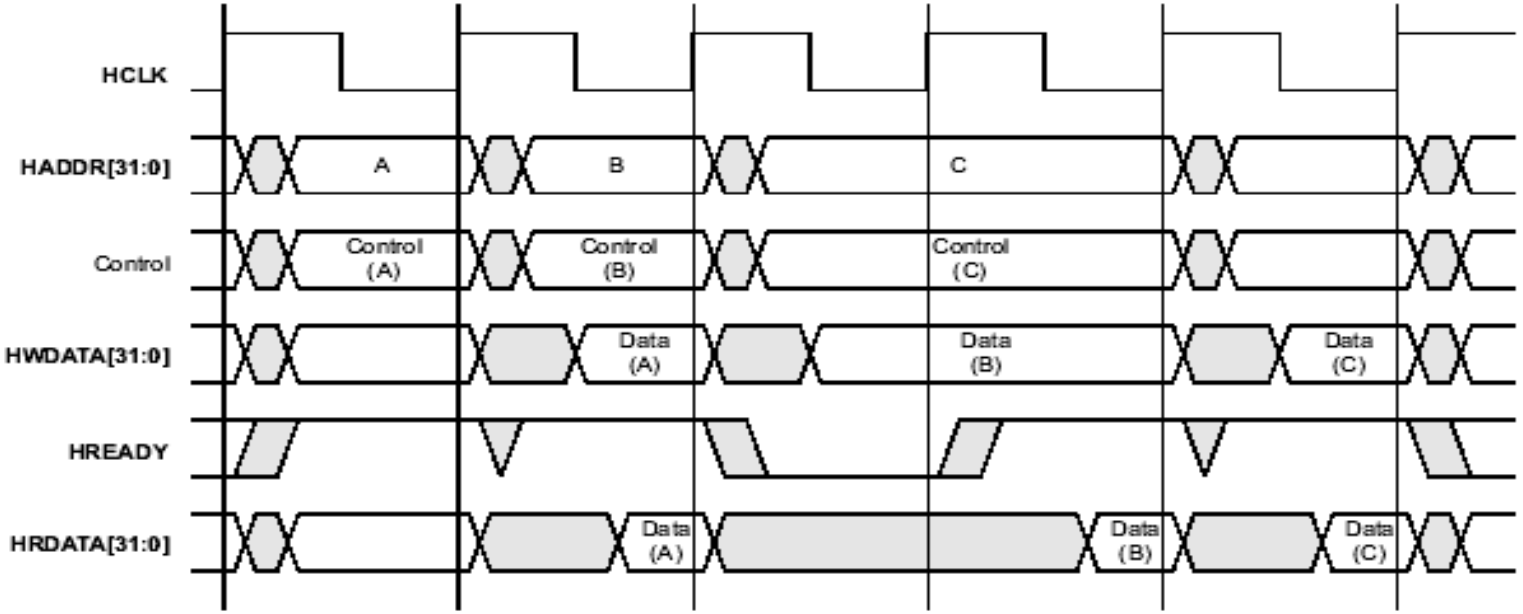
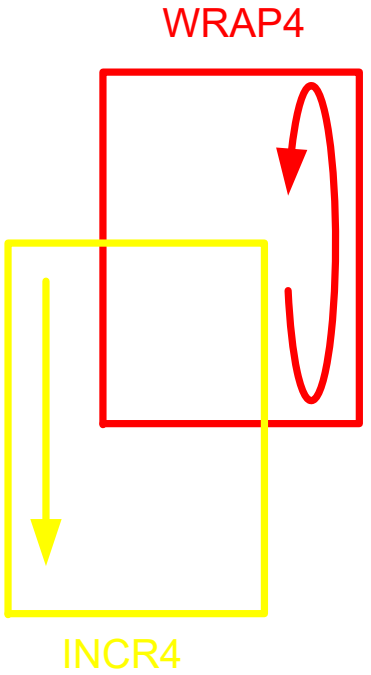


Figure 3-5 Multiple transfers

Bus operation (1/2)

- **A granted bus master starts an AHB transfer by driving address and control signals:**
  - **address**
  - **direction**
  - **width**
  - **burst forms**
    - **Incrementing burst:** not wrap at address boundaries
    - **Wrapping burst:** wrap at particular address boundaries



Address wrap in 4-word boundary

Pipelined Burst Transfers :Address phase

- During the address phase:
  - The master places the address and the other control signals on the bus
  - The decoder selects the appropriate slave
  - At the next rising edge of the clock, the slave stores the address and control signals
  - The data phase begins

Pipelined Burst Transfers: Address phase (continued)

- The address phase can be stretched if:
  - The bus is not immediately granted to the master (HGRANT = 0)
  - The preceding data phase is not yet complete (HREADY = 0)

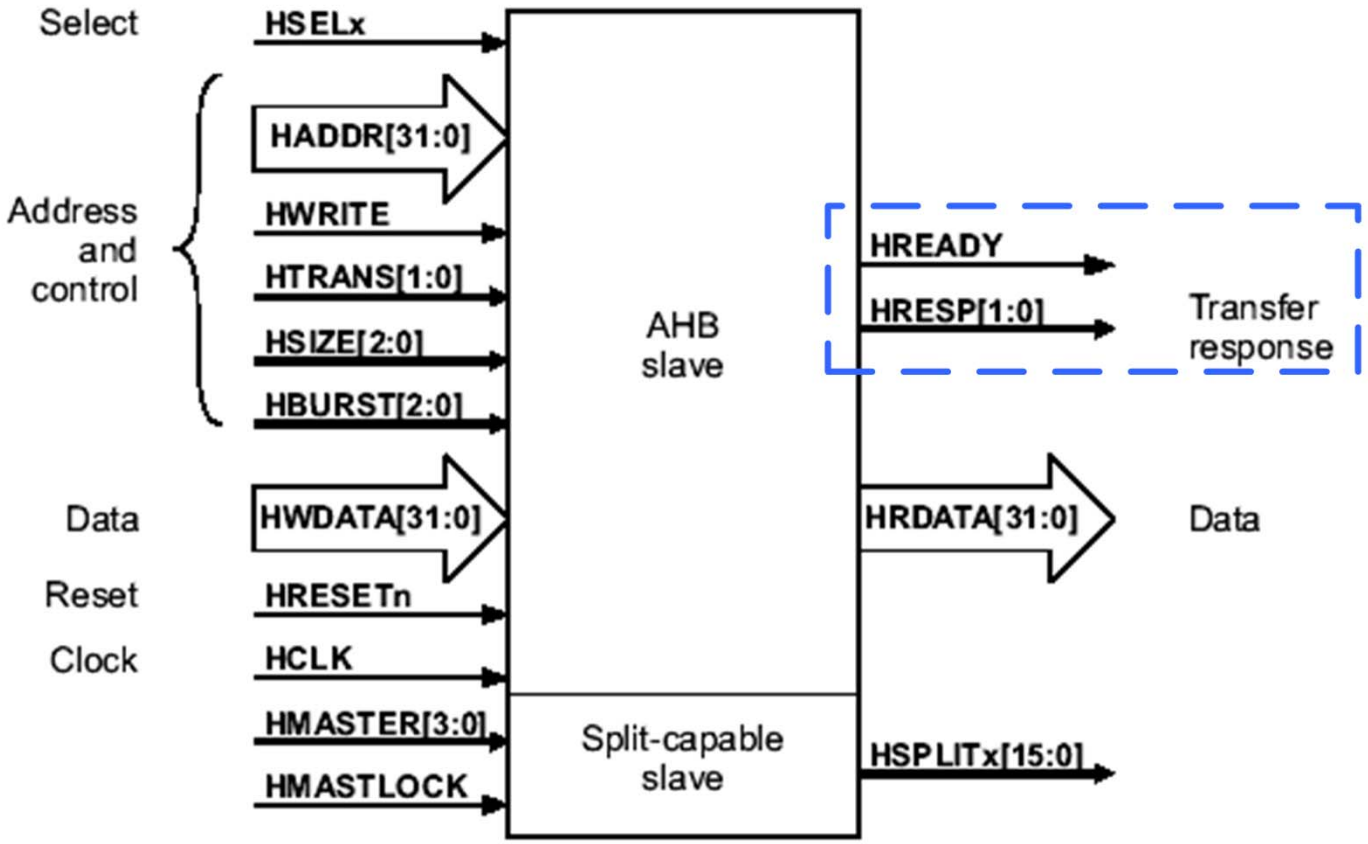
Pipelined Burst Transfers: Data phase

- During the data phase:
  - The slave reacts to the access, according to the stored control signals.
  - A slow slave can request several cycles by setting HREADY = 0.
  - When HREADY = 1, the access is complete. If the access is a read, the master captures the data at that moment.

Transfer Type

Type	HTRANS[1:0]	Descripton
IDLE	00	Slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer should be ignored by the slave.
BUSY	01	Masters cannot take next transfer place immediately during a burst transfer
NONSEQ	10	Indicates the first transfer of a burst or a single transfer
SEQ	11	The remaining transfers in a burst are SEQUENTIAL

Slave transfer responses(1/2)



Slave transfer responses(2/2)

- HREADY
- HRESP[1:0] Response

  00 OKAY

  01 ERROR

  10 RETRY

  11 SPLIT

- Two-cycle response

  – ERROR & RETRY & SPLIT

  – To complete current transfer, master can take following action

Bus operation (2/2)

- **During a transfer the slave shows the status using the response signals HRESP[1:0]**
  - **OKAY**: transfer progressing normally when HREADY is HIGH, transfer has completed successfully
  - **ERROR**: transfer error
  - **RETRY and SPLIT**: transfer can't complete immediately, but the bus master should continue to attempt the transfer

Requesting bus access

- Normally the arbiter will only grant different bus master when a burst is completing. However, if required, the arbiter can terminate a bust early to allow a higher priority master access.
- If the master requires locked accesses then it must also assert the **HLOCKx** signal to indicate to the arbiter that no other masters should be granted the bus.
- When a master is granted the bus and is performing a fixed length burst it is not necessary to continue to request the bus in order to complete the burst.
- It is possible that a master can be granted the bus when it is not requesting it.
  - if a master does not require access to the bus it drives the transfer type HTRANS to indicate an IDLE transfer.

Granting Bus Access

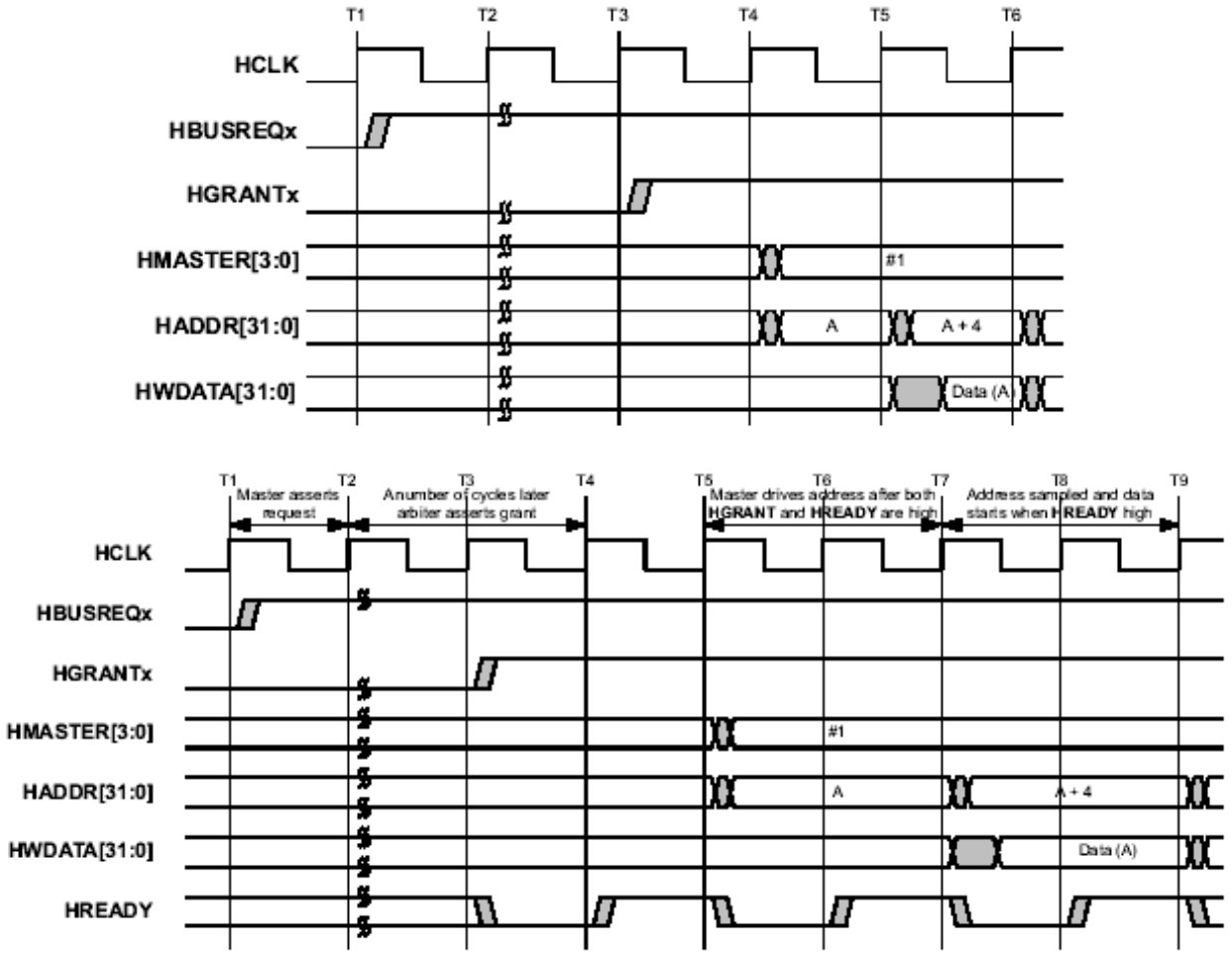


Figure 3-16 Granting access with wait states

The ownership of the data bus is delayed from the ownership of the address bus.

Handover After Burst